ROUTINE_FOR_OPERATION could be a "LOCK add shared-value, update-amount" bus lock add instruction.

[0039] The program unit 405 is an example translation of the program unit 401 in accordance with blocks 313 and 314 of Figure 3. The program unit 405 includes an atomic update routine as described in block 314. The atomic update routine call in the program unit 405 can have the following exemplary form:

atomic-update-routine(shared-value, update-amount, callback-routine);

[0040] A program unit 409 is an example of the atomic-update-routine called in the program unit 405. The atomic update routine illustrated in the program unit 409 includes a callback routine as described in block 313 of Figure 3. An example of the callback routine can be the following:

```
callback-routine(shared-value, update-amount)
{ new-shared-value = memory-update-operation(shared-value, update-amount);
  return(new-shared-value);
}
```

The callback routine encapsulates the memory update operation so that it can be called from the atomic update routine.

[0041] The implementation of the atomic update routine illustrated in the program unit 409 utilizes a CAS instruction to ensure atomicity of the atomic operation. The CAS instruction can be defined as follows:

```
CAS (compare, swap, lock)
{ if (compare == lock) then {lock = swap; return 0;}
  else { compare = lock; return 1;}
}
```

Various embodiments of the described invention can be implemented with other low-level instructions to ensure atomicity.

[0042] The program unit 407 is an example translation of the program unit 401 in accordance with block 315 of Figure 3. The program unit 407 includes a call to a lock-acquire routine and a call to a lock-release routine that surround the memory update operation. The lock routines in the program unit 407 are used to lock the shared memory location in order to update it with the result of the memory update operation. Locking the shared memory location prevents the shared memory location from being modified by a second thread while the memory update operation is being performed by a first thread. The lock routines can be optimized for portability by implementing separate lock routines for each supported platform with low-level instruction(s). The low-level instruction(s) can be fetch-and-add, bus lock instructions, TAS, etc. An alternative embodiment of the invention may implement platform independent lock routines with higher level instructions.

[0043] Figure 5 is a flow chart for performing the translated program unit 409 according to one embodiment of the invention. Figures 6A-6D are diagrams illustrating threads performing the program unit 409 according to one embodiment of the invention. Figure 5 will be described with reference to Figures 6A-6D and Figure 4. At block 501, a thread of a team arrives at the ATOMIC_UPDATE_ROUTINE of the program unit 405. At block 503, the thread loads the shared-value from a shared memory location to be updated into a memory location that has been allocated to the thread ("compare memory location").

[0044] Figure 6A is a diagram illustrating threads performing the first line of the atomic update routine in the program unit 409 of Figure 4 according to one embodiment of the invention. In Figure 6A, threads 603 and 605 arrive at the following line of the example ATOMIC_UPDATE_ROUTINE:

compare = shared-value;

In accordance with this line, the thread 603 loads the shared-value of a shared memory location 607 into a memory location 609 allocated to the thread 603. Also in response to this line, the thread 605 loads the shared-value of the shared memory location 607 into a memory location 615 that has been allocated to the thread 605.

[0045] Returning to Figure 5 at block 505, the thread performs the memory update operation with the compare-value. At block 507, the thread loads the result of the memory update operation into a second memory location that is allocated to the thread ("swap memory location") in accordance with the following line of the program unit 409:

swap = callback-routine (compare, update-amount);

[0046] Figure 6B is a diagram illustrating the threads 603 and 605 arriving at the callback routine in the program unit 409 according to one embodiment of the invention. In Figure 6B, the threads 603 and 605 arrive at the callback routine of the program unit 409. Both threads 603 and 605 perform a callback routine, such as the previous example of a callback routine. The threads 603 and 605 assign the result of the memory update operation to a new shared-value and load the new shared value into memory locations 611 and 617, respectively.

[0047] Returning to Figure 5 at block 509, the thread determines if the shared-value in the shared memory location is equal to the compare value loaded into the compare memory location. If these values are not equal, then at block 511 the thread indicates that the atomic operation has not been performed (e.g., sets an execute bit to false). At block 512, it is determined if the atomic operation is to be attempted again. If the thread is not to attempt the atomic operation again,. then at block 517 the thread is done. If the thread is to attempt the atomic operation again, then control flows from block 512 to block 503. If at block 509 the compare value and the shared-value are equal, then the shared-value has not been modified and the thread loads the swap value from the